```
// Computer Program Listing Appendix Under 37 CFR 1.52(e)
// Entitlement.java
// Copyright (c) 2004. Financial Fusion, Inc. All Rights Reserved.
package com.ffusion.entitlements;
import java.util.ArrayList;
/**
 This class represents a single Entitlement.
*/
public class Entitlement implements EntitlementCodes
{
 // Fields
 public static final String ENTITLEMENT  = "ENTITLEMENT";
 public static final String ID     = "ID";
 public static final String NAME    = "NAME";
 public static final String LIMITS    = "LIMITS";
 // Data Members
 private String userName;
 private String userType = String.valueOf(CUSTOMER_USER_TYPE);
 private String name;   // The name of the entitlement
 private String id;
 private ArrayList limits = new ArrayList();
 private EntitlementService entService;
 private Limit currentLimit;
 /**
  Constructs a new Entitlement object.
 */
 public Entitlement()
 {
 }
 /**
  Gets the ArrayList collection of Limits.
 */
 public ArrayList getLimits()
 {
  return limits;
 }
 /**
  * Sets currentLimit to the specified limit.
  * @param id the id of the limit.
  */
 public void setCurrentLimitById(String id)
 {
  this.currentLimit = this.getLimitById(id);
 }
 /**
  * Sets currentLimit to the specified limit.
  * @param name the name of the limit.
  */
 public void setCurrentLimitByName(String name)
 {
```

```
 this.currentLimit = this.getLimitByName(name);
}
/**
 * Gets the value of the current Limit object.
 * @return the limit value as a String.
 */
public String getCurrentLimitValue()
{
 if(this.currentLimit == null)
  return "";
 else
  return currentLimit.getLimitValue();
}
/**
 Add a Limit into the ArrayList collection.
 @param l the limit to add.
 @return whether the add was successful
*/
protected boolean addLimit(Limit l)
{
 if(l==null)
  return false;
 limits.add(l);
 return true;
}
/**
 Delete the Limit from the ArrayList collection.
 @param id the id of the Limit object.
 @return whether the delete was successful
*/
protected boolean deleteLimitById(String id)
{
 java.util.Iterator iterator = limits.iterator();
 while (iterator.hasNext())
 {
  Limit nextLimit = (Limit) iterator.next();
  if (nextLimit.getId().equals(id))
  {
   limits.remove(nextLimit);
   return true;
  }
 }
 return false;
}
/**
 Delete the Limit from the ArrayList collection.
 @param name name of the Limit object.
 @return whether the delete was successful
*/
protected boolean deleteLimitByName(String name)
```

```
{
 java.util.Iterator iterator = limits.iterator();
 while (iterator.hasNext())
 {
  Limit nextLimit = (Limit) iterator.next();
  if (nextLimit.getName().equals(name))
  {
   limits.remove (nextLimit);
   return true;
  }
 }
 return false;
}
/**
 Get the Limit from the ArrayList collection.
 @param id the id of the object.
*/
protected Limit getLimitById(String id)
{
 java.util.Iterator iterator = limits.iterator();
 while (iterator.hasNext())
 {
  Object o = iterator.next();
  if(o instanceof Limit)
  {
   if (((Limit)o).getId().equals(id))
    return (Limit)o;
  }
 }
 return null;
}
/**
 Get the Limit from the ArrayList collection.
 @param name the name of the Limit object.
*/
protected Limit getLimitByName(String name)
{
 int size = limits.size ();
 java.util.Iterator iterator = limits.iterator();
 while (iterator.hasNext())
 {
  Object o = iterator.next();
  if(o instanceof Limit)
  {
   if (((Limit)o).getName().equals(name))
    return (Limit)o;
  }
 }
 return null;
}
```

```
/**
 Set the Entitlement Service.  This service is used
 when the Entitlement object is checking to see if any Limits
 have been exceeded.  Running totals are checked and updated
 by using the service.
 @param service the Entitlement Service
*/
protected void setEntService(EntitlementService service)
{
 this.entService = service;
}
/**
 Set the name of the Entitlement.
 @param nameParm the name of the entitlement.
*/
public void setName(String nameParm)
{
 this.name = nameParm;
}
/**
 Get the name of the Entitlement.
 @return the name of the entitlement
*/
public String getName()
{
 return this.name;
}
/**
 Set the id of the Entitlement.
 @param ID the id of the entitlement
*/
public void setId(String ID)
{
 this.id = ID;
}
/**
 Get the id of the Entitlement.
 @return the id of the entitlement
*/
public String getId()
{
 return this.id;
}
/**
 Set the userName of who this Entitlement is associated
 with.
 @param nameParm the userName associated with this entitlement.
*/
public void setUserName(String nameParm)
{
```

```
 this.userName = nameParm;
}
/**
 Get the userName of who this Entitlement is associated
 with.
 @return the userName associated with this entitlement.
*/
public String getUserName()
{
 return this.userName;
}
/**
 Set the userType of who this Entitlement is associated
 with.
 @param type the userType associated with this entitlement.
*/
public void setUserType(String type)
{
 this.userType = type;
}
/**
 Get the userType of who this Entitlement is associated
 with.
 @return the userType associated with this entitlement.
*/
public String getUserType()
{
 return this.userType;
}
/**
 Creates and returns a copy of itself.
 @return an Entitlement object
*/
protected Entitlement copy()
{
 Entitlement retEnt = new Entitlement();
 retEnt.setName (this.getName ());
 Limit nextLimit = null;
 java.util.Iterator iterator = limits.iterator ();
 while (iterator.hasNext())
 {
  nextLimit = (Limit) iterator.next();
  retEnt.addLimit(nextLimit.copy ());
 }
 return retEnt;
}
/**
 Sets the Entitlement information in this object to
 the information in the Entitlement passed in.
 @param entitlement the Entitlement to use in setting.
```

```
*/
public void set(Entitlement entitlement)
{
 this.setId (entitlement.getId ());
 this.setUserName (entitlement.getUserName ());
 this.setUserType (entitlement.getUserType ());
 this.setName(entitlement.getName());
 this.limits = entitlement.limits;
}
/**
* Gets all Limits that have been exceeded and populates the
* exdLimits object passed.  This object is null if no limits have
* been exceeded.
* @param lmtbl the Limitable object to check.
* @param exdLimits the collection of limits that were exceed by the
*  Limitable object passed in.
*/
public int getExceededLimits(Limitable lmtbl, ArrayList exdLimits)
{
 if(this.entService == null)
  return this.ERROR_NO_ENTITLEMENT_SERVICE_IN_BEAN;
 int iRet = SUCCESS_CODE;
 iRet = entService.getRunningTotals (limits);
 Limit nextLimit = null;
 java.util.Iterator iterator = limits.iterator ();
 while (iterator.hasNext())
 {
  nextLimit = (Limit) iterator.next();
  if (nextLimit.isLimitExceeded (lmtbl))
  {
   exdLimits.add (nextLimit);
   iRet = this.ERROR_LIMITS_EXCEEDED;
  }
 }
 return iRet;
}
/**
* Updates the running totals of all limits associated with this
* Entitlement.
* @param lmtbl The ValueLimitable object to use in updating.
*/
protected int updateRunningTotals(ValueLimitable lmtbl)
{
 if(this.entService == null)
  return this.ERROR_NO_ENTITLEMENT_SERVICE_IN_BEAN;
 else if(!(lmtbl instanceof ValueLimitable))
  return this.SUCCESS_CODE;
 else
  return entService.updateRunningTotals(lmtbl,limits);
}
```

```
 /**
 * Rollsback the running totals of all limits associated with this
 * Entitlement.
 * @param lmtbl The ValueLimitable object to use in rolling back
 */
 protected int rollbackRunningTotals(ValueLimitable lmtbl)
 {
  if(this.entService == null)
   return this.ERROR_NO_ENTITLEMENT_SERVICE_IN_BEAN;
  int iRet = SUCCESS_CODE;
  iRet = entService.getRunningTotals (limits);
  if(iRet == SUCCESS_CODE)
   iRet = entService.rollbackRunningTotals(lmtbl,limits);
  return iRet;
 }
}
// EntitlementCodes.java
// Copyright (c) 2004. Financial Fusion, Inc. All Rights Reserved.
package com.ffusion.entitlements;
/**
 Constants for the entitlements package.
*/
public interface EntitlementCodes extends  java.io.Serializable
{
 /** The code for success.  Value is 0. */
 static final int SUCCESS_CODE = 0; // No Error occurred
 //Bean Codes
 /** Error - the user is not entitled.  Value is 14000. */
 static final int ERROR_NOT_ENTITLED         = 14000;
 /** Error - the user is entitled, but one or more limits have been exceeded.  Value is 14001. */
 static final int ERROR_LIMITS_EXCEEDED      = 14001;
 /** Error - the entitlement service is missing from a bean that requires one.  Value is 14002. */
 static final int ERROR_NO_ENTITLEMENT_SERVICE_IN_BEAN    = 14002;
 /** Error - an attempt was made to add a null to the limit collection in an entitlement.  Value is 14000. */
 static final int ERROR_NO_LIMIT_TO_ADD                   = 14003;
 // Task Codes
 /** Error - no entitlement service.  Value is 14100. */
 static final int ERROR_NO_ENTITLEMENT_SERVICE       = 14100;
 /** Error - the entitlements collection is not in the session.  Value is 14101. */
 static final int ERROR_NO_ENTITLEMENTS      = 14101;
 /** Error - an entitlements object is malformed.  Value is 14102. */
 static final int ERROR_MALFORMED_ENTITLEMENTS       = 14102;
 /** Error - the user is not entitled to perform the task, or an
 entitlement object is required for the task to process, and the object is missing.  Value is 14103. */
 static final int ERROR_NO_ENTITLEMENT       = 14103;
// static final int ERROR_NO_USER          = 14104;
 /** Error - a limit object is required for the task to process, and the object is missing.  Value is 14105. */
 static final int ERROR_NO_LIMIT         = 14105;
 /** Error - a limits collection is required for the task to process, and the object is missing.  Value is 14106. */
 static final int ERROR_NO_LIMITS        = 14106;
```

```
/** Error - an entitlement name is missing.  Value is 14107. */
static final int ERROR_NO_ENTITLEMENT_NAME       = 14107;
/** Error - the user is not entitled.  Value is 14108. */
static final int ERROR_NO_ENTITLEMENT_FOR_THIS_OPERATION    = 14108;
/** Error - when copying entitlements, source and target collections need to be specified.
  If none are found, this error is created.  Value is 14109. */
static final int ERROR_NO_TARGET_OR_SOURCE_ENTITLEMENTS     = 14109;
/** Error - the limit has no name.  Value is 14110. */
static final int ERROR_NO_LIMIT_NAME                        = 14110;
/** Error - the entitlement service name is required, but not provided.  Value is 14111. */
static final int ERROR_NO_ENTITLEMENT_SERVICE_NAME                  = 14111;
/** Error - the limit has no value.  Value is 14112. */
static final int ERROR_NO_LIMIT_VALUE        = 14112;
/** Error - a period type needs to be specified.  Value is 14113. */
static final int ERROR_NO_PERIOD_TYPE        = 14113;
/** Error - a period needs to be specified.  Value is 14114. */
static final int ERROR_NO_PERIOD         = 14114;
/** Error - no payee is specified when one is required.  Value is 14115. */
static final int ERROR_NO_PAYEE_ID         = 14115;
/** Error - no account is specified when one is required.  Value is 14116. */
static final int ERROR_NO_ACCOUNT_ID         = 14116;
/** Error - when validating a period limit, the period limit is determined to be malformed.  Value is 14117. */
static final int ERROR_MALFORMED_PERIOD_LIMIT      = 14117;
/** Error - when validating a transaction limit, the limit is determined to be malformed.  Value is 14118. */
static final int ERROR_MALFORMED_TRANSACTION_LIMIT     = 14118;
/** Error - when validating a Told limit, the limit is determined to be malformed.  Value is 14119. */
static final int ERROR_MALFORMED_TOID_LIMIT       = 14119;
/** Error - when validating a FromId limit, the limit is determined to be malformed.  Value is 14120. */
static final int ERROR_MALFORMED_FROMID_LIMIT      = 14120;
/** Error - a day is specified that is not valid.  Value is 14121. */
static final int ERROR_NOT_A_VALID_DAY        = 14121;
/** Error - a week is specified that is not valid.  Value is 14122. */
static final int ERROR_NOT_A_VALID_WEEK        = 14122;
/** Error - a month is specified that is not valid.  Value is 14123. */
static final int ERROR_NOT_A_VALID_MONTH       = 14123;
/** Error - a year is specified that is not valid.  Value is 14124. */
static final int ERROR_NOT_A_VALID_YEAR        = 14124;
/** Error - a period type is specified that is not valid.  Valid values include
DAILY, WEEKLY, MONTHLY, YEARLY.  Value is 14125. */
static final int ERROR_NOT_A_VALID_PERIOD_TYPE      = 14125;
/** Error - an entitlement is required for the task to process, but none was found.  Value is 14126. */
static final int ERROR_NO_ENTITLEMENT_FOUND       = 14126;
/** Error - a limit object is required for the task to process, but none was found.  Value is 14127. */
static final int ERROR_NO_LIMIT_FOUND         = 14127;
/** Error - an entitlements name is required but not specified.  Value is 14128. */
static final int ERROR_NO_ENTITLEMENTS_NAME                 = 14128;
/** Error - no user id was specified when one was required.  Value is 14129. */
static final int ERROR_NO_USER_ID                   = 14129;
/** Error - no user type was specified when one was required.  Value is 14130. */
static final int ERROR_NO_USER_TYPE                   = 14130;
```

```
 /** Error - the limit object was malformed.  Value is 14131. */
 static final int ERROR_MALFORMED_LIMIT                    = 14131;
 /** Error - the entitlement group id is invalid.  Value is 14132. */
 static final int ERROR_INVALID_GROUPID     = 14132;
 /** Error - the specified group type is invalid.  Value is 14133. */
 static final int ERROR_INVALID_GROUP_TYPE    = 14133;
 //Constants
 /** The minimum value for a day in the year.  Value is 1. */
 static final int MIN_DAY_IN_YEAR          = 1;
 /** The maximum value for a day in the year.  Value is 366. */
 static final int MAX_DAY_IN_YEAR          = 366;
 /** The minimum value for a month in the year.  Months are zero-based.  Value is 0. */
 static final int MIN_MONTH_IN_YEAR          = 0;
 /** The maximum value for a month in the year.  Months are zero-based.  Value is 11. */
 static final int MAX_MONTH_IN_YEAR          = 11;
 /** The minimum value for a week in the year.  Weeks are zero-based.  Value is 0. */
 static final int MIN_WEEK_IN_YEAR          = 0;
 /** The maximum value for a week in the year.  Value is 54. */
 static final int MAX_WEEK_IN_YEAR          = 54;
 /** The minimum value for a year value.  Value is 1. */
 static final int MIN_YEAR            = 1;
 /** The maximum value for a year value.  Value is 5000000. */
 static final int MAX_YEAR            = 5000000;
 //User type defines
 /** User types are used to differentiate between users that may have the
 same id, but are different types.  Customer User Type.  Value is 1. */
 static final int CUSTOMER_USER_TYPE          = 1;
 /** User types are used to differentiate between users that may have the
 same id, but are different types.  Employee User Type.  Value is 2. */
 static final int EMPLOYEE_USER_TYPE          = 2;
 /** User types are used to differentiate between users that may have the
 same id, but are different types.  Group User Type.  Value is 4. */
 static final int GROUP_USER_TYPE          = 4;
 //Common Limit values to validate bean information
 static final String LIMIT_ID       = "LIMIT_ID";
 static final String LIMIT_NAME       = "LIMIT_NAME";
 static final String LIMIT_VALUE       = "LIMIT_VALUE";
 static final String PERIOD       = "PERIOD";
 static final String PAYEEID       = "PAYEEID";
 static final String ACCOUNTID       = "ACCOUNTID";
}
// Entitlements.java
// Copyright (c) 2004. Financial Fusion, Inc. All Rights Reserved.
package com.ffusion.entitlements;
import java.util.ArrayList;
import java.util.HashMap;
/**
 This class represents a collection of Entitlements for a specific user.
 Entitlements are used to determine whether a particular task, or function may be performed.
*/
```

```
public class Entitlements extends ArrayList implements EntitlementCodes
{
 // Data Members
 private String userId;
 private String userType = String.valueOf(CUSTOMER_USER_TYPE);
 private ArrayList exceededLimits;
 private Entitlement currentEntitlement;
 private Limit currentLimit;
 private Entitlements usersEnts;
 private EntitlementService svc;
 private boolean initialize = true;
 private HashMap uniqueLimits = new HashMap();
 private static final String USER_ID     = "ID";
 private static final String USER_TYPE     = "TP";
 /**
  * Constructs a new Entitlements object for a given locale.
  */
 protected Entitlements()
 {
 }
 /**
  * Constructs a new Entitlements object for a given user.
  * @param id  the userid of the current user.
  * @param type  the usertype of the current user.
  * @param entSvc  the service used.
  * @param ents  the Entitlements of of the current user.
  * @return Entitlements object
  */
 public Entitlements(String id, String type, EntitlementService entSvc, Entitlements ents)
 {
  this.userId = id;
  this.userType = type;
  this.svc = entSvc;
  if(ents != null && id.equals (ents.getUserId ()) && type.equals(ents.getUserType ()))
   this.usersEnts = null;
  else
   this.usersEnts = ents;
 }
 /**
  * Overriding the ArrayList addAll method.  This method does nothing and returns false.
  * @param c  the collection to add.
  */
 public boolean addAll(java.util.Collection c)
 {
  return false;
 }
 /**
  * Overriding the ArrayList addAll method.  This method does nothing and returns false.
  * @param c  the collection to add.
  * @param index the index to add at.
```

```
 */
public boolean addAll(int index, java.util.Collection c)
{
 return false;
}
/**
 * Adds the Object o to the Array List.  If the object is not an Entitlement, this method
 * returns false.  Otherwise the method tries to add the object.
 * @param o  the object to add.
 */
public boolean add(Object o)
{
 if(o instanceof Entitlement)
 {
  ((Entitlement)o).setEntService(svc);
  //We let them add entitlements if the entitlements object is being
  //initialized, otherwise we check the validateAddEntitlement method
  if(initialize || validateAddEntitlement((Entitlement)o) == SUCCESS_CODE)
   return super.add (o);
 }
 return false;
}
/**
 * Adds the Object o to the Array List.  If the object is not an Entitlement, this method
 * returns false.  Otherwise the method tries to add the object.
 * @param index  the spot to add the object at.
 * @param o  the object to add.
 */
public void add(int index, Object o)
{
 if(o instanceof Entitlement)
 {
  ((Entitlement)o).setEntService(svc);
  //We let them add entitlements if the entitlements object is being
  //initialized, otherwise we check the validateAddEntitlement method
  if(initialize || validateAddEntitlement((Entitlement)o) == SUCCESS_CODE)
   super.add(index,o);
 }
}
/**
 * Removes the Object from the Array List by Name.  If the object is not an Entitlement, this method
 * returns false.  Otherwise the method tries to remove the object.
 * @param Name  the name of the Entitlement to remove.
 */
public boolean remove(String Name)
{
 return remove(get(Name));
}
/**
 * Removes the Object o from the Array List.  If the object is not an Entitlement, this method
```

```
 * returns false.  Otherwise the method tries to remove the object.
 * @param o  the object to remove.
 */
public boolean remove(Object o)
{
 if(o instanceof Entitlement)
 {
  if(initialize || validateEntitlement((Entitlement)o) == SUCCESS_CODE)
   return super.remove(o);
 }
 return false;
}
/**
 * Removes the Object from the Array List at the index i.
 * @param i  the index of the object to remove.
 */
public Object remove(int i)
{
 return super.remove(i);
}
/**
 Delete the Limit from the ArrayList collection.
 @param name name of the Limit object.
*/
public int deleteLimit(String limitName, String entName)
{
 int errorCode = validateDeleteLimit(limitName,entName);
 if(errorCode == SUCCESS_CODE)
 {
  Entitlement AdminEnt = get(entName);
  AdminEnt.deleteLimitByName(limitName);
 }
 return errorCode;
}
/**
 Delete the Limit from the ArrayList collection.
 @param name name of the Limit object.
*/
public int modifyLimit(Limit lim, String entName)
{
 int errorCode = validateAddLimit(lim,entName);
 if(errorCode == SUCCESS_CODE)
 {
  Limit l = (Limit)uniqueLimits.get(lim.getId ());
  if(l == null && validateAddLimit(lim, entName) == SUCCESS_CODE)
  {
   deleteLimit(lim.getName(), entName);
   addLimit(lim, get(entName));
  }
  else
```

```
   l.set (lim);
  }
  return errorCode;
 }
 private void modifyLimit(Limit lim, Entitlement ent)
 {
  //Limit limitToAdd = lim.copy ();
  Limit l = ent.getLimitByName(lim.getName ());
  if(l == null)
   ent.addLimit(lim);
  else
  {
   ent.deleteLimitByName(lim.getName());
   lim.setId(l.getId ());
   ent.addLimit(lim);
  }
 }
 private int validateEntitlement(Entitlement e)
 {
  //If they are trying to modify their own entitlements, we fail.
  if(usersEnts == null)
   return ERROR_NOT_ENTITLED;
  // If entitlement name is null or empty, we fail.
  if((e.getName() == null) || (e.getName().equals("")))
   return ERROR_NO_ENTITLEMENT_NAME;
  //We return if we passed in nulls
  if(e==null)
   return ERROR_NO_ENTITLEMENT;
  //If the entitlement does not exist in the user's collection, we fail
  if(!isUserEntitled(e.getName()))
   return ERROR_NOT_ENTITLED;
  return SUCCESS_CODE;
 }
 private int validateAddEntitlement(Entitlement e)
 {
  int errorCode = validateEntitlement(e);
  if(errorCode != SUCCESS_CODE)
   return errorCode;
  Entitlement UserEnt = usersEnts.get(e.getName());
  //Now check to make sure that any limits the user's entitlement has,
  //is not exceeded and that any limits the user has is added to the list
  ArrayList lims = UserEnt.getLimits();
  for(int i = 0; i < lims.size (); i++)
  {
   Limit userLimit = (Limit)lims.get(i);
   Limit existingLimit = e.getLimitByName(userLimit.getName());
   if(existingLimit != null)
   {
    if(userLimit.isLimitExceeded(existingLimit))
     return ERROR_NOT_ENTITLED;
```

```
 }
}
 for(int i = 0; i < lims.size (); i++)
 {
  Limit userLimit = (Limit)lims.get(i);
  Limit existingLimit = e.getLimitByName(userLimit.getName());
  if(existingLimit != null)
   modifyLimit(existingLimit, e.getName());
  else
  {
   Limit copy = userLimit.copy();
   addLimit(copy, e);
  }
 }
 return errorCode;
}
private int validateDeleteLimit(String limName, String entName)
{
 int errorCode = SUCCESS_CODE;
 //If they are trying to modify their own entitlements, we fail.
 if(usersEnts == null)
  return ERROR_NOT_ENTITLED;
 //We return if we passed in nulls
 if(limName==null)
  return ERROR_NO_LIMIT_NAME;
 if(entName == null)
  return ERROR_NO_ENTITLEMENT_NAME;
 //Get the entitlement from this collection
 errorCode = isEntitled(entName);
 if(errorCode != SUCCESS_CODE)
  return ERROR_NOT_ENTITLED;
 Entitlement UserEnt = usersEnts.get(entName);
 if(UserEnt != null && UserEnt.getLimitByName(limName) != null)
  return ERROR_NOT_ENTITLED;
 else
  return SUCCESS_CODE;
}
private int validateAddLimit(Limit l, String entName)
{
 int errorCode = SUCCESS_CODE;
 //If they are trying to modify their own entitlements, we fail.
 if(usersEnts == null)
  return ERROR_NOT_ENTITLED;
 //We return if we passed in nulls
 if(l==null)
  return ERROR_NO_LIMIT_TO_ADD;
 if(entName == null)
  return ERROR_NO_ENTITLEMENT_NAME;
 if (l.getName() == null || l.getName().equals(""))
  return ERROR_NO_LIMIT_NAME;
```

```
 //Get the entitlement from this collection
 errorCode = isEntitled(entName);
 if(errorCode != SUCCESS_CODE)
  return ERROR_NOT_ENTITLED;
 //If the entitlement does not exist in the user's collection, we fail
 if(!isUserEntitled(entName))
  return ERROR_NOT_ENTITLED;
 Entitlement UserEnt = usersEnts.get(entName);
 //Now check to make sure that any limits the user's entitlement has,
 //is not exceeded.
 Limit UserLim = UserEnt.getLimitByName(l.getName ());
 if(UserLim != null && UserLim.isLimitExceeded(l))
  return ERROR_LIMITS_EXCEEDED;
 else
  return SUCCESS_CODE;
}
/**
 Add a Limit into the Entitlement specified.
 @param l the limit to add.
 @param entName the name of the Entitlement to add to.
*/
public int addLimit(Limit l, String entName)
{
 int errorCode = validateAddLimit(l, entName);
 //We let them add limits, if the entitlements object is being
 //initialized, otherwise we check the validateAddLimit method
 if(initialize || errorCode == SUCCESS_CODE)
 {
  Entitlement adminEnt = get(entName);
  addLimit(l, adminEnt);
 }
 return errorCode;
}
private void addLimit(Limit l, Entitlement ent)
{
 if (l != null && ent != null)
 {
  modifyLimitId(l);
  if(uniqueLimits.get(l.getId ()) != null)
   l = (Limit)uniqueLimits.get(l.getId ());
  else
   uniqueLimits.put(l.getId (),l);
  //Now just add the limit
  ent.addLimit(l);
 }
}
private void modifyLimitIds()
{
 Limit l = null;
 java.util.Iterator it = uniqueLimits.values ().iterator ();
```

```
 while(it.hasNext ())
 {
  Object o = it.next();
  if(o instanceof Limit)
  {
   modifyLimitId((Limit)o);
  }
 }
}
private void modifyLimitIds(Entitlement ent)
{
 Limit l = null;
 java.util.Iterator it = ent.getLimits ().iterator ();
 while(it.hasNext ())
 {
  Object o = it.next();
  if(o instanceof Limit)
  {
   modifyLimitId((Limit)o);
  }
 }
}
private void modifyLimitId(Limit l)
{
 String id = l.getId ();
 if(id.indexOf(USER_ID) != -1)
  id = id.substring(0,id.indexOf(USER_ID));
 id = id + USER_ID + this.getUserId () + USER_TYPE + this.getUserType ();
 l.setId(id);
 if(uniqueLimits.get(id) == null)
  uniqueLimits.put(id, l);
}
/**
 Makes the Entitlements object passed in a copy of itself.
    @param ents  the object that will be the copy.
*/
public void copy(Entitlements ents)
{
 Object o = null;
 Entitlement e = null;
 ents.clear();
 java.util.Iterator it = this.iterator ();
 while(it.hasNext ())
 {
  o = it.next ();
  if(o instanceof Entitlement)
   ents.add(((Entitlement)o).copy());
 }
}
/**
```

```
 * Gets the user id for whose entitlements are contained in this collection.
 * @return the userid.
 */
public String getUserId()
{
 return userId;
}
/**
 * Gets the user type for whose entitlements are contained in this collection.
 * @return the user type.
 */
public String getUserType()
{
 return userType;
}
/**
 * Gets the Entitlement.
 * @param name  the name of the Entitlement.
 * @return the Entitlement.
 */
public Entitlement get(String name)
{
 if(name == null)
  return null;
 java.util.Iterator iterator = iterator();
 while (iterator.hasNext())
 {
  Entitlement nextEnt = (Entitlement) iterator.next();
  if (name.equals(nextEnt.getName ()))
  {
   return nextEnt;
  }
 }
 return null;
}
/**
 * Gets the Entitlement.
 * @param id  the id of the Entitlement.
 * @return the Entitlement.
 */
public Entitlement getById(String id)
{
 if(id == null)
  return null;
 java.util.Iterator iterator = iterator();
 while (iterator.hasNext())
 {
  Entitlement nextEnt = (Entitlement) iterator.next();
  if (nextEnt.getId ().equals (id))
  {
```

```
   return nextEnt;
  }
 }
 return null;
}
/**
 Get the Limit from the ArrayList collection.
 @param entName the name of the entitlement.
 @param limitId the id of the Limit.
 @return the Limit requested
*/
public Limit getLimitById(String entName, String limitId)
{
 Entitlement ent = get(entName);
 if(ent!=null)
  return ent.getLimitById(limitId);
 else
  return null;
}
/**
 Get the Limit from the ArrayList collection.
 @param entName the name of the entitlement.
 @param limitName the name of the Limit.
 @return the Limit requested
*/
public Limit getLimitByName(String entName, String limitName)
{
 Entitlement ent = get(entName);
 if(ent!=null)
  return ent.getLimitByName(limitName);
 else
  return null;
}
/**
 * Returns whether the entitlement exists and whether any limits are broken.
 * If entitlement exists and no limits are broken, then Running Totals are
 * updated.  This should be called immediately before a TransactionLimitable object
 * executes its transaction.
 * @param EntitlementName  the name of the entitlement to check.
 * @param lmtbl  the Limitable object to check against.
 * @return integer of whether or not it succeeded 0--means SUCCESS.
 */
public int isEntitled(String EntitlementName, Limitable lmtbl)
{
 int iret = SUCCESS_CODE;
 this.exceededLimits = new ArrayList();
 Entitlement ent = this.get(EntitlementName);
 if (ent == null)
  iret = this.ERROR_NOT_ENTITLED;
 else
```

```
 {
  iret = ent.getExceededLimits (lmtbl, this.exceededLimits);
  int sizeofLimits = this.exceededLimits.size ();
  if(iret == SUCCESS_CODE && lmtbl instanceof ValueLimitable)
   iret = ent.updateRunningTotals((ValueLimitable)lmtbl);
 }
 return iret;
}
/**
 * Rolls back the running total update that was made when isEntitled is called.  Normal
 * processing in a Limitable object is to have the call to isEntitled be made immediately before
 * the Limitable's transaction.  If isEntitled returns true, then the transaction is processed.
 * If the transaction fails, then this method is called to rollback the running total call.
 * @param EntitlementName  the name of the entitlement that was checked.
 * @param lmtbl  the Limitable object that was checked.
 * @return integer of whether or not it succeeded 0--means SUCCESS.
 */
public int rollbackRunningTotals(String EntitlementName, Limitable lmtbl)
{
 int iret = SUCCESS_CODE;
 Entitlement ent = this.get(EntitlementName);
 if (ent == null)
  iret = this.ERROR_NOT_ENTITLED;
 else if(lmtbl instanceof ValueLimitable)
  iret = ent.rollbackRunningTotals((ValueLimitable)lmtbl);
 return iret;
}
private boolean isUserEntitled(String EntitlementName)
{
 return (usersEnts.get(EntitlementName) != null);
}
/**
 * Returns whether the entitlement exists in this this Entitlements collection.
 * @param EntitlementName  the name of the Entitlement.
 * @return integer of whether or not it exists 0--means exists.
 */
public int isEntitled(String EntitlementName)
{
 Entitlement ent = this.get(EntitlementName);
 if (ent == null)
  return this.ERROR_NOT_ENTITLED;
 else
  return this.SUCCESS_CODE;
}
/**
 * Sets the current entitlement object for the entitlement whose
 * name is passed in.
 * @param name  the name of the Entitlement to set as current.
 */
public void setCurrentEntitlement(String name)
```

```
{
 this.currentEntitlement = null;
 if(name == null)
  return;
 java.util.Iterator iterator = iterator();
 while (iterator.hasNext())
 {
  Entitlement nextEnt = (Entitlement) iterator.next();
  if (nextEnt.getName ().equals (name))
  {
   this.currentEntitlement = nextEnt;
   return;
  }
 }
}
/**
 * Gets whether the current entitlement exists
 * @return boolean of whether the entitlement exists or not.
 */
public boolean getCurrentEntitlement()
{
 if(this.currentEntitlement != null)
  return true;
 else
  return false;
}
/**
 * Sets the current limit object for the limit whose
 * name is passed in.
 * @param name  the name of the limit to set as current.
 */
public void setCurrentLimit(String name)
{
 this.currentLimit = null;
 if(name == null || this.currentEntitlement == null)
  return;
 this.currentLimit = this.currentEntitlement.getLimitByName (name);
}
/**
 * Gets the value of the current limit
 * return String the Value of the current limit.
 */
public String getCurrentLimitValue()
{
 if(this.currentLimit != null && this.currentLimit.getLimitValue () != null)
  return this.currentLimit.getLimitValue ();
 else
  return "";
}
/**
```

```
 * Gets the exceeded limits of a failed entitlement search
 * @return ArrayList of the Limits that were exceeded by the most recent call to isEntitled.
 */
public ArrayList getExceededLimits()
{
 return this.exceededLimits;
}
/**
 * Sets the initialize boolean to false.  This boolean is used to determine
 * whether or not the Entitlements object is initializing.  This allows the
 * service implementation to add to the object, and then to set this flag
 * disallowing illigitimate modifications.
 */
public void setInitialize()
{
 this.initialize = false;
}
}
// FromIdLimit.java
// Copyright (c) 2004. Financial Fusion, Inc. All Rights Reserved.
package com.ffusion.entitlements;
import java.util.Locale;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.*;
import java.util.HashMap;
/**
 * This class contains a list of all ids that are not entitled for the task.
 */
public class FromIdLimit implements Limit, EntitlementCodes
{
 // Data members
 private String id;
 private String name;
 private String limitValue;
 /**
  * Constructs a new FromIdLimit object.
 */
 public FromIdLimit()
 {
 }
 /**
 * Gets the list of FromIds.
 * @return Collection containing the FromIds that are excluded from the Entitlement.
 */
 public ArrayList getFromIds()
 {
  ArrayList lst = new ArrayList();
  StringTokenizer parser = new StringTokenizer(limitValue,",");
  while(parser.hasMoreTokens ())
```

```
 {
  String str = (String)parser.nextToken ();
  lst.add(str);
 }
 return lst;
}
/**
* Sets the list of FromIds.
* @param FromIds  the list of FromIds that are excluded from the Entitlement.
*/
public void setFromIds(ArrayList FromIds)
{
 boolean bfirst = true;
 StringBuffer sbuf = new StringBuffer();
 java.util.Iterator iterator = FromIds.iterator ();
 if(iterator.hasNext ())
 {
  if(!bfirst)
   sbuf.append(",");
  bfirst = false;
  Object nextVal = iterator.next();
  sbuf.append(nextVal.toString ());
 }
 this.limitValue = sbuf.toString ();
}
/**
* Sets the id property.
* @param sId  the id of the limit.
*/
public void setId(String sId)
{
 this.id = sId;
}
/**
* Gets the id property.
* @return the id of the limit.
*/
public String getId()
{
 return this.id;
}
/**
* Sets the name property.
* @param sName  the Name of the limit.
*/
public void setName(String sName)
{
 this.name = sName;
}
/**
```

```
* Gets the name property.
* @return the Name of the limit.
*/
public String getName()
{
 return this.name;
}
/**
* Gets the limitValue associated with this Limit.
* @return the Limit Value.
*/
public String getLimitValue()
{
 return limitValue;
}
/**
* Sets the limitValue associated with this Limit.
* @param  the Limit Value.
*/
public void setLimitValue(String s)
{
 limitValue = s;
}
/**
 Creates and returns a copy of itself.
 @return a Limit object
*/
public Limit copy()
{
 FromIdLimit retLim = new FromIdLimit();
 retLim.setId (this.getId ());
 retLim.setName (this.getName ());
 retLim.setLimitValue (this.getLimitValue ());
 return retLim;
}
/**
 sets the Limit information in this object based on the Limit passed in
 @param lim object passed in to use in the set.
*/
public void set(Limit lim)
{
 if(lim != null && lim instanceof FromIdLimit)
 {
  this.setId (lim.getId ());
  this.setName (lim.getName ());
  this.setLimitValue (lim.getLimitValue ());
 }
}
/**
 Checks to see whether the Limit object has limit values that exceed the Limit.
```

```
 @param limit the object to check the limit against.
 @return Whether the limit was exceeded.
*/
public boolean isLimitExceeded(Limit limit)
{
 if(limit instanceof FromIdLimit)
 {
  FromIdLimit lim = (FromIdLimit)limit;
  String checkLimitValue = lim.getLimitValue();
  String value;
  String checkValue;
  boolean bFound = false;
  StringTokenizer userParser = new StringTokenizer(limitValue,",");
  while(userParser.hasMoreTokens ())
  {
   bFound = false;
   value = (String)userParser.nextToken ();
   StringTokenizer parser = new StringTokenizer(checkLimitValue,",");
   while(parser.hasMoreTokens ())
   {
    checkValue = (String)parser.nextToken ();
    if(checkValue.equals(value))
     bFound = true;
   }
   //If we can't find this, the limit is exceeded
   if(!bFound)
    return true;
  }
 }
 return false;
}
/**
 Checks to see if the object coming in is going to exceed the limit.
 @param lmtbl  the object to be tested
 @return Whether this limit was exceeded
*/
public boolean isLimitExceeded(Limitable limitable)
{
 if (limitable instanceof FromIdLimitable)
 {
  FromIdLimitable limit = (FromIdLimitable) limitable;
  String FromId = limit.getLimitableFromID();
  //if there is no limitable id, we will return false
  if(FromId == null)
   return false;
  StringTokenizer parser = new StringTokenizer(limitValue,",");
  while(parser.hasMoreTokens ())
  {
   String value = (String)parser.nextToken ();
   // If the FromId is found in this limit, the limit is exceeded.
```

```
    // Thus, the task cannot be performed.
    if (FromId.compareTo(value) == 0)
     return true;
   }
  }
  return false;
 }
 /**
  Gets the Name Value pairs of the Limit
  @return the Name/Value pairs.
 */
 public HashMap getNameValuePairs()
 {
  HashMap map = new HashMap();
  if(this.getLimitValue () != null)
  {
   map.put("LimitValue",this.getLimitValue ());
  }
  return map;
 }
}
// Limit.java
// Copyright (c) 2004. Financial Fusion, Inc. All Rights Reserved.
package com.ffusion.entitlements;
import java.util.HashMap;
/**
 An object is a Limit if it implements this interface.
*/
public interface Limit extends java.io.Serializable
{
 /**
  Checks to see whether the Limitable object would exceed the Limit.
  @param limitable the object to check the limit against.
  @return Whether the limit was exceeded.
 */
 public boolean isLimitExceeded(Limitable limitable);
 /**
  Checks to see whether the Limit object has limit values that exceed the Limit.
  @param limit the object to check the limit against.
  @return Whether the limit was exceeded.
 */
 public boolean isLimitExceeded(Limit limit);
 /**
  Creates and returns a copy of itself.
  @return a Limit object
 */
 public Limit copy();
 /**
  Gets the Id of the Limit
  @return the Id.
```

```
 */
 public String getId();
 /**
  Sets the Id of the Limit
  @param id the Id.
 */
 public void setId(String id);
 /**
  Gets the Name of the Limit
  @return the Name.
 */
 public String getName();
 /**
  Sets the Name of the Limit
  @param sName the Name.
 */
 public void setName(String sName);
 /**
  Gets the Limit Value of the Limit
  @return the Limit Value.
 */
 public String getLimitValue();
 /**
  Gets the Name Value pairs of the Limit
  @return the Name/Value pairs.
 */
 public HashMap getNameValuePairs();
 /**
  sets the Limit information in this object based on the Limit passed in
  @param lim object passed in to use in the set.
 */
 public void set(Limit lim);
}
// PeriodLimit.java
// Copyright (c) 2004. Financial Fusion, Inc. All Rights Reserved.
package com.ffusion.entitlements;
import java.util.Locale;
import java.text.*;
import java.util.Calendar;
import java.util.HashMap;
/**
 This class represents a single PeriodLimit.
*/
public class PeriodLimit implements Limit, EntitlementCodes
{
 /** The DAILY static variable is used to specify Daily Period Limits.  Value is "DAILY". */
 public static final String DAILY    = "DAILY";
 /** The WEEKLY static variable is used to specify Weekly Period Limits.  Value is "WEEKLY". */
 public static final String WEEKLY    = "WEEKLY";
 /** The MONTHLY static variable is used to specify Monthly Period Limits.  Value is "MONTHLY". */
```

```
public static final String MONTHLY     = "MONTHLY";
/** The YEARLY static variable is used to specify Yearly Period Limits.  Value is "YEARLY". */
public static final String YEARLY    = "YEARLY";
// Data Members
private String id;
private String name;
private String periodType = DAILY;   // Type of period--Defaults to DAILY
private int period = Calendar.SUNDAY;  // period--Defaults to Sunday
private String accountId;     // Unique ID of the account
private String payeeId;      // Unique ID of the payee
private double runningTotal;
private double limit;      // Limit to compare
/**
 Constructs a new PeriodLimit object
*/
public PeriodLimit()
{
}
/**
 Set the periodType property
 @param typeParm  the periodType of the periodLimit
*/
public void setPeriodType(String typeParm)
{
 this.periodType = typeParm;
}
/**
 Get the periodType property--this defaults to DAILY
 @return periodType
*/
public String getPeriodType()
{
 return this.periodType;
}
/**
 Set the period property
 @param periodParm  the period of the periodLimit
*/
public void setPeriod(String periodParm)
{
 try{
  this.period = Integer.parseInt (periodParm);
 }
 catch( NumberFormatException e ){
  this.period = 0;
 }
}
/**
 Get the period property--This is the numeric representation of the period.
 Numeric values may be viewed in EntitlementCodes as MIN_DAY_IN_YEAR...
```

```
 @return period
*/
public String getPeriod()
{
 return Integer.toString (period);
}
/**
 Set the accountId property
 @param idParm  the accountId of the periodLimit
*/
public void setAccountID(String idParm)
{
 this.accountId = idParm;
}
/**
 Get the accountId property
 @return accountId
*/
public String getAccountID()
{
 return this.accountId;
}
/**
 Set the payeeId property
 @param idParm  the payeeId of the periodLimit
*/
public void setPayeeID(String idParm)
{
 this.payeeId = idParm;
}
/**
 Get the payeeId property
 @return payeeId
*/
public String getPayeeID()
{
 return this.payeeId;
}
/**
 Set the limit property
 @param Limit  the limit of the periodLimit
*/
public void setLimitValue(String Limit)
{
 try{
  this.limit = Double.parseDouble (Limit);
 }
 catch( NumberFormatException e ){
 }
}
```

```
/**
 Creates and returns a copy of itself.
 @return a Limit object
*/
public Limit copy()
{
 PeriodLimit retLim = new PeriodLimit();
 retLim.setId (this.getId ());
 retLim.setAccountID (this.getAccountID ());
 retLim.setPayeeID (this.getPayeeID ());
 retLim.setLimitValue (this.getLimitValue ());
 retLim.setName (this.getName ());
 retLim.setPeriod (this.getPeriod ());
 retLim.setPeriodType (this.getPeriodType ());
 return retLim;
}
/**
 sets the Limit information in this object based on the Limit passed in
 @param lim object passed in to use in the set.
*/
public void set(Limit lim)
{
 if(lim != null && lim instanceof PeriodLimit)
 {
  PeriodLimit periodLimit = (PeriodLimit)lim;
  this.setAccountID (periodLimit.getAccountID ());
  this.setId (periodLimit.getId ());
  this.setLimitValue (periodLimit.getLimitValue ());
  this.setName (periodLimit.getName ());
  this.setPayeeID (periodLimit.getPayeeID ());
  this.setPeriod (periodLimit.getPeriod ());
  this.setPeriodType (periodLimit.getPeriodType ());
  this.setRunningTotalValue(periodLimit.getRunningTotalValue ());
 }
}
/**
* Sets the id property.
* @param sId  the id of the limit.
*/
public void setId(String sId)
{
 this.id = sId;
}
/**
* Gets the id property.
* @return the id of the limit.
*/
public String getId()
{
 return this.id;
```

```
}
/**
* Sets the name property.
* @param sName  the Name of the limit.
*/
public void setName(String sName)
{
 this.name = sName;
}
/**
* Gets the name property.
* @return the Name of the limit.
*/
public String getName()
{
 return this.name;
}
/**
* Gets the limitValue associated with this Limit.
* @return the Limit Value.
*/
public String getLimitValue()
{
 return String.valueOf(this.limit);
}
/**
* Sets the runningTotal property.
* @param s  the runningTotal.
*/
public void setRunningTotal(String s)
{
 this.runningTotal = Double.parseDouble(s);
}
/**
* Gets the runningTotal property.
* @return the running total for this limit.
*/
public String getRunningTotal()
{
 return String.valueOf(this.runningTotal);
}
/**
* Sets the runningTotal property.
* @param d  the runningTotal.
*/
public void setRunningTotalValue(double d)
{
 this.runningTotal = d;
}
/**
```

```
* Gets the runningTotal property.
* @return the running total for this limit as a double.
*/
public double getRunningTotalValue()
{
 return this.runningTotal;
}
/**
 Checks to see whether the Limit object has limit values that exceed the Limit.
 @param limit the object to check the limit against.
 @return Whether the limit was exceeded.
*/
public boolean isLimitExceeded(Limit checkLim)
{
 if(checkLim instanceof PeriodLimit)
 {
  PeriodLimit lim = (PeriodLimit)checkLim;
  if(!lim.getPeriodType().equals(getPeriodType ()))
   return false;
  try
  {
   if(Double.valueOf(lim.getLimitValue()).doubleValue () > limit )
    return true;
  }
  catch(Exception e)
  {
   return false;
  }
 }
 return false;
}
/**
 Checks to see if the object coming in is going to exceed the limit.
 @param lmtbl  the object to be tested
 @return boolean value of whether this limit was exceeded
*/
public boolean isLimitExceeded(Limitable lmtbl)
{
 boolean bret = false;
 boolean bCheck = true;
 //If this is not a TransactionLimitable object, we won't check
 if(lmtbl instanceof TransactionLimitable)
 {
  TransactionLimitable txnLmt = (TransactionLimitable) lmtbl;
  //If we are returning nulls out of this, we will return false
  if(this.accountId != null && txnLmt.getLimitableFromID() == null)
   return false;
  if(this.payeeId != null && txnLmt.getLimitableToID() == null)
   return false;
  if(txnLmt.getLimitableValue() == null)
```

```
    return false;
   //if we have an account, and they are not equal, we don't need to check
   if(this.accountId != null && !(txnLmt.getLimitableFromID ().equals (this.accountId)))
    bCheck = false;
   //if we have an account, and they are not equal, we don't need to check
   if(bCheck && this.payeeId != null && !(txnLmt.getLimitableToID ().equals (this.payeeId)))
    bCheck = false;
   //If we are supposed to check, and the limit is exceeded, we note that
   if(bCheck)
   {
    if(limit < (txnLmt.getLimitableValue ().doubleValue () + runningTotal))
     bret = true;
   }
  }
  return bret;
 }
 /**
  Gets the Name Value pairs of the Limit
  @return the Name/Value pairs.
 */
 public HashMap getNameValuePairs()
 {
  HashMap map = new HashMap();
  if(this.getLimitValue () != null)
   map.put("LimitValue",this.getLimitValue ());
  if(this.getAccountID () != null)
   map.put("AccountID",this.getAccountID ());
  if(this.getPayeeID () != null)
   map.put("PayeeID",this.getPayeeID ());
  if(this.getPeriodType () != null)
   map.put("PeriodType",this.getPeriodType ());
  if(this.getId () == null || this.getId ().length () == 0)
  {
   map.put("RunningTotal",getRunningTotal());
   map.put("Period",getPeriod());
  }
  return map;
 }
}
// ToldLimit.java
// Copyright (c) 2004. Financial Fusion, Inc. All Rights Reserved.
package com.ffusion.entitlements;
import java.util.Locale;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.*;
import java.util.HashMap;
/**
 * This class contains a list of all ids that are not entitled for the task.
 */
```

```
public class ToIdLimit implements Limit, EntitlementCodes
{
 // Data members
 private String id;
 private String name;
 private String limitValue;
 /**
  * Constructs a new ToIdLimit object.
 */
 public ToIdLimit()
 {
 }
 /**
 * Gets the list of ToIds.
 * @return ArrayList containing the ToIds that are excluded from the Entitlement.
 */
 public ArrayList getToIds()
 {
  ArrayList lst = new ArrayList();
  StringTokenizer parser = new StringTokenizer(limitValue,",");
  while(parser.hasMoreTokens ())
  {
   String str = (String)parser.nextToken ();
   lst.add(str);
  }
  return lst;
 }
 /**
 * Sets the list of ToIds.
 * @param ToIds the list of ToIds that are excluded from the Entitlement.
 */
 public void setToIds(ArrayList ToIds)
 {
  boolean bfirst = true;
  StringBuffer sbuf = new StringBuffer();
  java.util.Iterator iterator = ToIds.iterator ();
  if(iterator.hasNext ())
  {
   if(!bfirst)
    sbuf.append(",");
   bfirst = false;
   Object nextVal = iterator.next();
   sbuf.append(nextVal.toString ());
  }
  this.limitValue = sbuf.toString ();
 }
 /**
 * Sets the id property.
 * @param sId the id of the limit.
 */
```

```
public void setId(String sId)
{
 this.id = sId;
}
/**
* Gets the id property.
* @return the id of the limit.
*/
public String getId()
{
 return this.id;
}
/**
* Sets the name property.
* @param sName the Name of the limit.
*/
public void setName(String sName)
{
 this.name = sName;
}
/**
* Gets the name property.
* @return the Name of the limit.
*/
public String getName()
{
 return this.name;
}
/**
* Gets the limitValue associated with this Limit.
* @return the Limit Value.
*/
public String getLimitValue()
{
 return limitValue;
}
/**
* Sets the limitValue associated with this Limit.
* @param s the Limit Value.
*/
public void setLimitValue(String s)
{
 limitValue = s;
}
/**
 Creates and returns a copy of itself.
 @return a Limit object
*/
public Limit copy()
{
```

```
 ToldLimit retLim = new ToldLimit();
 retLim.setId (this.getId ());
 retLim.setName (this.getName ());
 retLim.setLimitValue (this.getLimitValue ());
 return retLim;
}
/**
 sets the Limit information in this object based on the Limit passed in
 @param lim object passed in to use in the set.
*/
public void set(Limit lim)
{
 if(lim != null && lim instanceof ToldLimit)
 {
  this.setId (lim.getId ());
  this.setName (lim.getName ());
  this.setLimitValue (lim.getLimitValue ());
 }
}
/**
 Checks to see whether the Limit object has limit values that exceed the Limit.
 @param limit the object to check the limit against.
 @return Whether the limit was exceeded.
*/
public boolean isLimitExceeded(Limit limit)
{
 if(limit instanceof ToldLimit)
 {
  ToldLimit lim = (ToldLimit)limit;
  String checkLimitValue = lim.getLimitValue();
  String value;
  String checkValue;
  boolean bFound = false;
  StringTokenizer userParser = new StringTokenizer(limitValue,",");
  while(userParser.hasMoreTokens ())
  {
   bFound = false;
   value = (String)userParser.nextToken ();
   StringTokenizer parser = new StringTokenizer(checkLimitValue,",");
   while(parser.hasMoreTokens ())
   {
    checkValue = (String)parser.nextToken ();
    if(checkValue.equals(value))
     bFound = true;
   }
   //If we can't find this, the limit is exceeded
   if(!bFound)
    return true;
  }
 }
```

```
 return false;
}
/**
 Checks to see if the object coming in is going to exceed the limit.
 @param lmtbl the object to be tested
 @return boolean value of whether this limit was exceeded
*/
public boolean isLimitExceeded(Limitable limitable)
{
 if (limitable instanceof ToIdLimitable)
 {
  ToIdLimitable limit = (ToIdLimitable) limitable;
  String ToId = limit.getLimitableToID();
  //if there is no limitable id, we will return false
  if(ToId == null)
   return false;
  StringTokenizer parser = new StringTokenizer(limitValue,",");
  while(parser.hasMoreTokens ())
  {
   String value = (String)parser.nextToken ();
   // If the ToId is found in this limit, the limit is exceeded.
   // Thus, the task cannot be performed.
   if (ToId.compareTo(value) == 0)
    return true;
  }
 }
 return false;
}
/**
 Gets the Name Value pairs of the Limit
 @return the Name/Value pairs.
*/
public HashMap getNameValuePairs()
{
 HashMap map = new HashMap();
 if(this.getLimitValue () != null)
 {
  map.put("LimitValue",this.getLimitValue ());
 }
 return map;
}
}
// TransactionLimit.java
// Copyright (c) 2004. Financial Fusion, Inc. All Rights Reserved.
package com.ffusion.entitlements;
import java.util.Locale;
import java.text.*;
import java.util.HashMap;
/**
 This class represents a single TransactionLimit.
```

```
*/
public class TransactionLimit implements Limit, EntitlementCodes
{
 // Data Members
 private String id;
 private String name;
 protected String accountId;    // Unique ID of the account
 protected String payeeId;    // Unique ID of the payee
 protected double limit;     // Limit to compare
 /**
  Constructs a new TransactionLimit object
 */
 public TransactionLimit()
 {
 }
 /**
  Set the accountId property
  @param idParm the accountId of the transactionLimit
 */
 public void setAccountID(String idParm)
 {
  this.accountId = idParm;
 }
 /**
  Get the accountId property
  @return accountId
 */
 public String getAccountID()
 {
  return this.accountId;
 }
 /**
  Set the payeeId property
  @param idParm the payeeId of the transactionLimit
 */
 public void setPayeeID(String idParm)
 {
  this.payeeId = idParm;
 }
 /**
  Get the payeeId property
  @return payeeId
 */
 public String getPayeeID()
 {
  return this.payeeId;
 }
 /**
  Set the limit property
  @param the limit of the transactionLimit
```

```
*/
public void setLimitValue(String Limit)
{
 try{
  this.limit = Double.parseDouble (Limit);
 }
 catch( NumberFormatException e ){
 }
}
/**
* Gets the limitValue associated with this Limit.
* @return The Limit Value.
*/
public String getLimitValue()
{
 return String.valueOf(this.limit);
}
/**
 Creates and returns a copy of itself.
 @return a Limit object
*/
public Limit copy()
{
 TransactionLimit retLim = new TransactionLimit();
 retLim.setId (this.getId ());
 retLim.setAccountID (this.getAccountID ());
 retLim.setPayeeID (this.getPayeeID ());
 retLim.setLimitValue (this.getLimitValue ());
 retLim.setName (this.getName ());
 return retLim;
}
/**
 sets the Limit information in this object based on the Limit passed in
 @param lim object passed in to use in the set.
*/
public void set(Limit lim)
{
 if(lim != null && lim instanceof TransactionLimit)
 {
  TransactionLimit transactionLimit = (TransactionLimit)lim;
  this.setId (transactionLimit.getId ());
  this.setName (transactionLimit.getName ());
  this.setAccountID (transactionLimit.getAccountID());
  this.setPayeeID (transactionLimit.getPayeeID());
  this.setLimitValue (transactionLimit.getLimitValue());
 }
}
/**
 Checks to see whether the Limit object has limit values that exceed the Limit.
 @param limit the object to check the limit against.
```

```
 @return Whether the limit was exceeded.
*/
public boolean isLimitExceeded(Limit checkLim)
{
 if(checkLim instanceof TransactionLimit)
 {
  TransactionLimit lim = (TransactionLimit)checkLim;
  try
  {
   if(Double.valueOf(lim.getLimitValue()).doubleValue () > limit )
    return true;
  }
  catch(Exception e)
  {
   return false;
  }
 }
 return false;
}
/**
 Checks to see if the object coming in is going to exceed the limit.
 @param lmtbl the object to be tested
 @return boolean value of whether this limit was exceeded
*/
public boolean isLimitExceeded(Limitable lmtbl)
{
 boolean bret = false;
 boolean bCheck = true;
 double d = this.limit;
 //If this is not a TransactionLimitable object, we won't check
 if(lmtbl instanceof TransactionLimitable)
 {
  TransactionLimitable txnLmt = (TransactionLimitable) lmtbl;
  //If we are returning nulls out of this, we will return false
  if(this.accountId != null && txnLmt.getLimitableFromID() == null)
   return false;
  if(this.payeeId != null && txnLmt.getLimitableToID() == null)
   return false;
  if(txnLmt.getLimitableValue() == null)
   return false;
  //if we have an account, and they are not equal, we don't need to check
  if(this.accountId != null && !(txnLmt.getLimitableFromID ().equals (this.accountId)))
   bCheck = false;
  //if we have an account, and they are not equal, we don't need to check
  if(bCheck && this.payeeId != null && !(txnLmt.getLimitableToID ().equals (this.payeeId)))
   bCheck = false;
  //If we are supposed to check, and the limit is exceeded, we note that
  if(bCheck && (this.limit < txnLmt.getLimitableValue ().doubleValue ()))
   bret = true;
 }
```

```
 return bret;
}
/**
 sets the TransactionLimit information in this object based on the TransactionLimit passed in
 @param transactionLimit - contact that TransactionLimit settings
public int update(Limitable lmtbl)
{
 return this.SUCCESS;
}
*/
/**
* Sets the id property.
* @param sId the id of the limit.
*/
public void setId(String sId)
{
 this.id = sId;
}
/**
* Gets the name property.
* @return The Name of the limit.
*/
public String getId()
{
 return this.id;
}
/**
* Sets the name property.
* @param sName the Name of the limit.
*/
public void setName(String sName)
{
 this.name = sName;
}
/**
* Gets the name property.
* @return The Name of the limit.
*/
public String getName()
{
 return this.name;
}
/**
 Gets the Name Value pairs of the Limit
 @return the Name/Value pairs.
*/
public HashMap getNameValuePairs()
{
 HashMap map = new HashMap();
 if(this.getLimitValue () != null)
```

```
   map.put("LimitValue",this.getLimitValue ());
  if(this.getAccountID () != null)
   map.put("AccountID",this.getAccountID ());
  if(this.getPayeeID () != null)
   map.put("PayeeID",this.getPayeeID ());
  return map;
 }
}
```